Data description

MSSubClass: Identifies the type of dwelling involved in the sale.

```
        20	1-STORY 1946 & NEWER ALL STYLES
        30	1-STORY 1945 & OLDER
        40	1-STORY W/FINISHED ATTIC ALL AGES
        45	1-1/2 STORY - UNFINISHED ALL AGES
        50	1-1/2 STORY FINISHED ALL AGES
        60	2-STORY 1946 & NEWER
        70	2-STORY 1945 & OLDER
        75	2-1/2 STORY ALL AGES
        80	SPLIT OR MULTI-LEVEL
        85	SPLIT FOYER
        90	DUPLEX - ALL STYLES AND AGES
       120	1-STORY PUD (Planned Unit Development) - 1946 & NEWER
       150	1-1/2 STORY PUD - ALL AGES
       160	2-STORY PUD - 1946 & NEWER
       180	PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
       190	2 FAMILY CONVERSION - ALL STYLES AND AGES
```

MSZoning: Identifies the general zoning classification of the sale.

```
       A	Agriculture
       C	Commercial
       FV	 Floating Village Residential
       I	Industrial
       RH	 Residential High Density
       RL	 Residential Low Density
       RP	 Residential Low Density Park
       RM	 Residential Medium Density
```

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

```
Grvl    Gravel
Pave    Paved
```

Alley: Type of alley access to property

```
Grvl    Gravel
Pave    Paved
NA     No alley access
```

LotShape: General shape of property

```
Reg    Regular
IR1    Slightly irregular
IR2    Moderately Irregular
IR3    Irregular
```

LandContour: Flatness of the property

```
Lvl    Near Flat/Level
Bnk    Banked - Quick and significant rise from street grade to building
HLS    Hillside - Significant slope from side to side
Low    Depression
```

Utilities: Type of utilities available

```
AllPub    All public Utilities (E,G,W,& S)
NoSewr    Electricity, Gas, and Water (Septic Tank)
NoSeWa    Electricity and Gas Only
ELO    Electricity only
```

LotConfig: Lot configuration

```
Inside    Inside lot
Corner    Corner lot
CulDSac    Cul-de-sac
FR2    Frontage on 2 sides of property
FR3    Frontage on 3 sides of property
```

LandSlope: Slope of property

```
Gtl    Gentle slope
Mod    Moderate Slope
Sev    Severe Slope
```

Neighborhood: Physical locations within Ames city limits

```
Blmngtn    Bloomington Heights
Blueste    Bluestem
BrDale    Briardale
BrkSide    Brookside
ClearCr    Clear Creek
CollgCr    College Creek
Crawfor    Crawford
Edwards    Edwards
Gilbert    Gilbert
IDOTRR    Iowa DOT and Rail Road
MeadowV    Meadow Village
Mitchel    Mitchell
Names    North Ames
NoRidge    Northridge
NPkVill    Northpark Villa
NridgHt    Northridge Heights
NWAmes    Northwest Ames
OldTown    Old Town
SWISU    South & West of Iowa State University
Sawyer    Sawyer
SawyerW    Sawyer West
Somerst    Somerset
StoneBr    Stone Brook
Timber    Timberland
Veenker    Veenker
```

Condition1: Proximity to various conditions

```
Artery    Adjacent to arterial street
Feedr    Adjacent to feeder street
Norm    Normal
RRNn    Within 200' of North-South Railroad
RRAn    Adjacent to North-South Railroad
PosN    Near positive off-site feature--park, greenbelt, etc.
PosA    Adjacent to postive off-site feature
RRNe    Within 200' of East-West Railroad
RRAe    Adjacent to East-West Railroad
```

Condition2: Proximity to various conditions (if more than one is present)

```
Artery    Adjacent to arterial street
Feedr    Adjacent to feeder street
Norm    Normal
RRNn    Within 200' of North-South Railroad
RRAn    Adjacent to North-South Railroad
PosN    Near positive off-site feature--park, greenbelt, etc.
PosA    Adjacent to postive off-site feature
RRNe    Within 200' of East-West Railroad
RRAe    Adjacent to East-West Railroad
```

BldgType: Type of dwelling

```
1Fam    Single-family Detached
2FmCon    Two-family Conversion; originally built as one-family dwelling
Duplx    Duplex
TwnhsE    Townhouse End Unit
TwnhsI    Townhouse Inside Unit
```

HouseStyle: Style of dwelling

```
1Story    One story
1.5Fin    One and one-half story: 2nd level finished
1.5Unf    One and one-half story: 2nd level unfinished
2Story    Two story
2.5Fin    Two and one-half story: 2nd level finished
2.5Unf    Two and one-half story: 2nd level unfinished
SFoyer    Split Foyer
SLvl    Split Level
```

OverallQual: Rates the overall material and finish of the house

```
10    Very Excellent
9    Excellent
8    Very Good
7    Good
6    Above Average
5    Average
4    Below Average
3    Fair
2    Poor
1    Very Poor
```

OverallCond: Rates the overall condition of the house

```
10    Very Excellent
9    Excellent
8    Very Good
7    Good
6    Above Average
5    Average
4    Below Average
3    Fair
2    Poor
1    Very Poor
```

YearBuilt: Original construction date

YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)

RoofStyle: Type of roof

```
Flat	Flat
Gable	Gable
Gambrel	Gabrel (Barn)
Hip	Hip
Mansard	Mansard
Shed	Shed
```

RoofMatl: Roof material

```
ClyTile	Clay or Tile
CompShg	Standard (Composite) Shingle
Membran	Membrane
Metal	Metal
Roll	Roll
Tar&Grv	Gravel & Tar
WdShake	Wood Shakes
WdShngl	Wood Shingles
```

Exterior1st: Exterior covering on house

```
AsbShng    Asbestos Shingles
AsphShn    Asphalt Shingles
BrkComm    Brick Common
BrkFace    Brick Face
CBlock    Cinder Block
CemntBd    Cement Board
HdBoard    Hard Board
ImStucc    Imitation Stucco
MetalSd    Metal Siding
Other    Other
Plywood    Plywood
PreCast    PreCast
Stone    Stone
Stucco    Stucco
VinylSd    Vinyl Siding
Wd Sdng    Wood Siding
WdShing    Wood Shingles
```

Exterior2nd: Exterior covering on house (if more than one material)

```
AsbShng    Asbestos Shingles
AsphShn    Asphalt Shingles
BrkComm    Brick Common
BrkFace    Brick Face
CBlock    Cinder Block
CemntBd    Cement Board
HdBoard    Hard Board
ImStucc    Imitation Stucco
MetalSd    Metal Siding
Other    Other
Plywood    Plywood
PreCast    PreCast
Stone    Stone
Stucco    Stucco
VinylSd    Vinyl Siding
Wd Sdng    Wood Siding
WdShing    Wood Shingles
```

MasVnrType: Masonry veneer type

```
BrkCmn    Brick Common
BrkFace    Brick Face
CBlock    Cinder Block
None    None
Stone    Stone
```

MasVnrArea: Masonry veneer area in square feet

ExterQual: Evaluates the quality of the material on the exterior

```
Ex    Excellent
Gd    Good
TA    Average/Typical
Fa    Fair
Po    Poor
```

ExterCond: Evaluates the present condition of the material on the exterior

```
Ex    Excellent
Gd    Good
TA    Average/Typical
Fa    Fair
Po    Poor
```

Foundation: Type of foundation

```
BrkTil    Brick & Tile
CBlock    Cinder Block
PConc    Poured Contrete
Slab    Slab
Stone    Stone
Wood    Wood
```

BsmtQual: Evaluates the height of the basement

```
Ex    Excellent (100+ inches)
Gd    Good (90-99 inches)
TA    Typical (80-89 inches)
Fa    Fair (70-79 inches)
Po    Poor (<70 inches
NA    No Basement
```

BsmtCond: Evaluates the general condition of the basement

```
Ex    Excellent
Gd    Good
TA    Typical - slight dampness allowed
Fa    Fair - dampness or some cracking or settling
Po    Poor - Severe cracking, settling, or wetness
NA    No Basement
```

BsmtExposure: Refers to walkout or garden level walls

```
Gd    Good Exposure
Av    Average Exposure (split levels or foyers typically score average or above)
Mn    Mimimum Exposure
No    No Exposure
NA    No Basement
```

BsmtFinType1: Rating of basement finished area

```
GLQ    Good Living Quarters
ALQ    Average Living Quarters
BLQ    Below Average Living Quarters
Rec    Average Rec Room
LwQ    Low Quality
Unf    Unfinshed
NA    No Basement
```

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types)

```
GLQ    Good Living Quarters
ALQ    Average Living Quarters
BLQ    Below Average Living Quarters
Rec    Average Rec Room
LwQ    Low Quality
Unf    Unfinshed
NA    No Basement
```

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

```
Floor    Floor Furnace
GasA    Gas forced warm air furnace
GasW    Gas hot water or steam heat
Grav    Gravity furnace
OthW    Hot water or steam heat other than gas
Wall    Wall furnace
```

HeatingQC: Heating quality and condition

```
Ex    Excellent
Gd    Good
TA    Average/Typical
Fa    Fair
Po    Poor
```

CentralAir: Central air conditioning

```
N    No
Y    Yes
```

Electrical: Electrical system

```
SBrkr    Standard Circuit Breakers & Romex
FuseA    Fuse Box over 60 AMP and all Romex wiring (Average)
FuseF    60 AMP Fuse Box and mostly Romex wiring (Fair)
FuseP    60 AMP Fuse Box and mostly knob & tube wiring (poor)
Mix    Mixed
```

1stFlrSF: First Floor square feet

2ndFlrSF: Second floor square feet

LowQualFinSF: Low quality finished square feet (all floors)

GrLivArea: Above grade (ground) living area square feet

BsmtFullBath: Basement full bathrooms

BsmtHalfBath: Basement half bathrooms

FullBath: Full bathrooms above grade

HalfBath: Half baths above grade

Bedroom: Bedrooms above grade (does NOT include basement bedrooms)

Kitchen: Kitchens above grade

KitchenQual: Kitchen quality

```
Ex    Excellent
Gd    Good
TA    Typical/Average
Fa    Fair
Po    Poor
```

TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

Functional: Home functionality (Assume typical unless deductions are warranted)

```
Typ    Typical Functionality
Min1    Minor Deductions 1
Min2    Minor Deductions 2
Mod    Moderate Deductions
Maj1    Major Deductions 1
Maj2    Major Deductions 2
Sev    Severely Damaged
Sal    Salvage only
```

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

```
Ex    Excellent - Exceptional Masonry Fireplace
Gd    Good - Masonry Fireplace in main level
TA    Average - Prefabricated Fireplace in main living area or Masonry Fireplace in basement
Fa    Fair - Prefabricated Fireplace in basement
Po    Poor - Ben Franklin Stove
NA    No Fireplace
```

GarageType: Garage location

```
2Types    More than one type of garage
Attchd    Attached to home
Basment    Basement Garage
BuiltIn    Built-In (Garage part of house - typically has room above garage)
CarPort    Car Port
Detchd    Detached from home
NA    No Garage
```

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

```
Fin    Finished
RFn    Rough Finished
Unf    Unfinished
NA    No Garage
```

GarageCars: Size of garage in car capacity

GarageArea: Size of garage in square feet

GarageQual: Garage quality

```
       Ex	Excellent
       Gd	Good
       TA	Typical/Average
       Fa	Fair
       Po	Poor
       NA	No Garage
```

GarageCond: Garage condition

```
       Ex	Excellent
       Gd	Good
       TA	Typical/Average
       Fa	Fair
       Po	Poor
       NA	No Garage
```

PavedDrive: Paved driveway

```
       Y	Paved 
       P	Partial Pavement
       N	Dirt/Gravel
```

WoodDeckSF: Wood deck area in square feet

OpenPorchSF: Open porch area in square feet

EnclosedPorch: Enclosed porch area in square feet

3SsnPorch: Three season porch area in square feet

ScreenPorch: Screen porch area in square feet

PoolArea: Pool area in square feet

PoolQC: Pool quality

```
Ex    Excellent
Gd    Good
TA    Average/Typical
Fa    Fair
NA    No Pool
```

Fence: Fence quality

```
GdPrv    Good Privacy
MnPrv    Minimum Privacy
GdWo    Good Wood
MnWw    Minimum Wood/Wire
```

Data Cleaning

In [196]:
```
import pandas as pd
```

In [197]:
```
df = pd.read_csv('house-prices-advanced-regression-techniques/train.csv')
df
```

Out[197]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	GdPrv
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN

1460 rows × 81 columns

In [198]:
```
df.isna().sum()
```

Out[198]:
```
Id                 0
MSSubClass         0
MSZoning           0
LotFrontage      259
LotArea            0
                ... 
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
SalePrice          0
Length: 81, dtype: int64
```

In [199]:
```
df.columns
```

Out[199]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [200]:
```
df['YrSold'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 1460 entries, 0 to 1459
Series name: YrSold
Non-Null Count  Dtype
--------------  -----
1460 non-null   int64
dtypes: int64(1)
memory usage: 11.5 KB
```

In [201]:
```
df['YrSold'].value_counts()
```

Out[201]:
```
2009    338
2007    329
2006    314
2008    304
2010    175
Name: YrSold, dtype: int64
```

In [202]:
```
df.YrSold[: 10]
```

Out[202]:
```
0    2008
1    2007
2    2008
3    2006
4    2008
5    2009
6    2007
7    2009
8    2008
9    2008
Name: YrSold, dtype: int64
```

In [203]:
```
# To check important columns

missing_percentage = df.isnull().mean() * 100
```

In [204]:
```
missing_percentage
```

Out[204]:
```
Id                0.000000
MSSubClass        0.000000
MSZoning          0.000000
LotFrontage      17.739726
LotArea           0.000000
                   ...
MoSold            0.000000
YrSold            0.000000
SaleType          0.000000
SaleCondition     0.000000
SalePrice         0.000000
Length: 81, dtype: float64
```

In [205]:
```
# sorted_columns
sorted_columns = missing_percentage.sort_values(ascending = False)
```

In [206]:
```
sorted_columns
```

Out[206]:
```
PoolQC          99.520548
MiscFeature     96.301370
Alley           93.767123
Fence           80.753425
FireplaceQu     47.260274
                  ...
ExterQual        0.000000
Exterior2nd      0.000000
Exterior1st      0.000000
RoofMatl         0.000000
SalePrice        0.000000
Length: 81, dtype: float64
```

In [207]:
```
for column, percentage in sorted_columns.items():
    print(f'{column}: {percentage: .2f}% missing')
```

```
PoolQC:  99.52% missing
MiscFeature:  96.30% missing
Alley:  93.77% missing
Fence:  80.75% missing
FireplaceQu:  47.26% missing
LotFrontage:  17.74% missing
GarageYrBlt:  5.55% missing
GarageCond:  5.55% missing
GarageType:  5.55% missing
GarageFinish:  5.55% missing
GarageQual:  5.55% missing
BsmtFinType2:  2.60% missing
BsmtExposure:  2.60% missing
BsmtQual:  2.53% missing
BsmtCond:  2.53% missing
BsmtFinType1:  2.53% missing
MasVnrArea:  0.55% missing
MasVnrType:  0.55% missing
Electrical:  0.07% missing
Id:  0.00% missing
Functional:  0.00% missing
Fireplaces:  0.00% missing
KitchenQual:  0.00% missing
KitchenAbvGr:  0.00% missing
BedroomAbvGr:  0.00% missing
HalfBath:  0.00% missing
FullBath:  0.00% missing
BsmtHalfBath:  0.00% missing
TotRmsAbvGrd:  0.00% missing
GarageCars:  0.00% missing
GrLivArea:  0.00% missing
GarageArea:  0.00% missing
PavedDrive:  0.00% missing
WoodDeckSF:  0.00% missing
OpenPorchSF:  0.00% missing
EnclosedPorch:  0.00% missing
3SsnPorch:  0.00% missing
ScreenPorch:  0.00% missing
PoolArea:  0.00% missing
MiscVal:  0.00% missing
MoSold:  0.00% missing
YrSold:  0.00% missing
SaleType:  0.00% missing
```

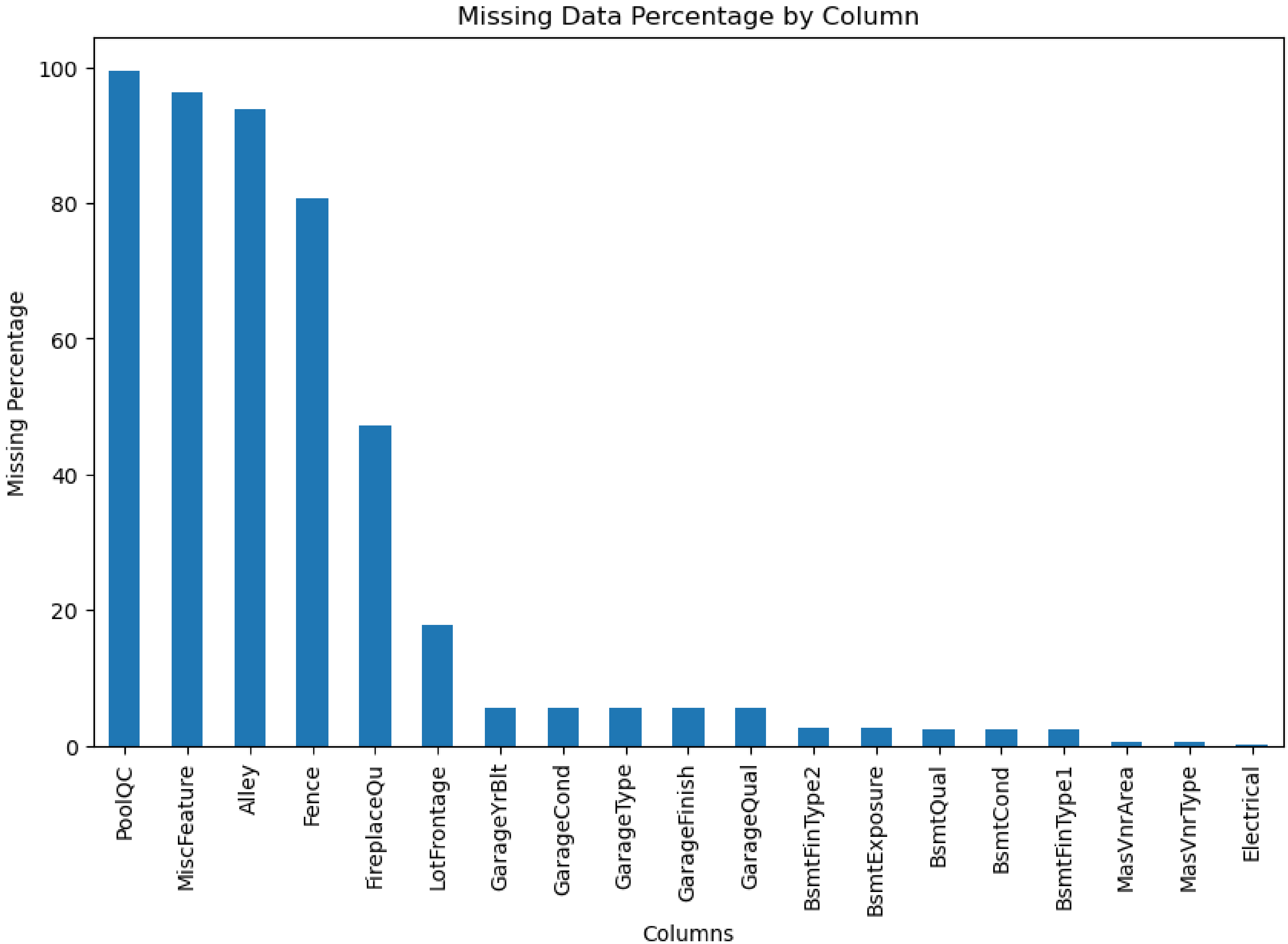
```
SaleCondition:  0.00% missing
BsmtFullBath:  0.00% missing
HeatingQC:  0.00% missing
LowQualFinSF:  0.00% missing
LandSlope:  0.00% missing
OverallQual:  0.00% missing
HouseStyle:  0.00% missing
BldgType:  0.00% missing
Condition2:  0.00% missing
Condition1:  0.00% missing
Neighborhood:  0.00% missing
LotConfig:  0.00% missing
YearBuilt:  0.00% missing
Utilities:  0.00% missing
LandContour:  0.00% missing
LotShape:  0.00% missing
Street:  0.00% missing
LotArea:  0.00% missing
MSZoning:  0.00% missing
OverallCond:  0.00% missing
YearRemodAdd:  0.00% missing
2ndFlrSF:  0.00% missing
BsmtFinSF2:  0.00% missing
1stFlrSF:  0.00% missing
CentralAir:  0.00% missing
MSSubClass:  0.00% missing
Heating:  0.00% missing
TotalBsmtSF:  0.00% missing
BsmtUnfSF:  0.00% missing
BsmtFinSF1:  0.00% missing
RoofStyle:  0.00% missing
Foundation:  0.00% missing
ExterCond:  0.00% missing
ExterQual:  0.00% missing
Exterior2nd:  0.00% missing
Exterior1st:  0.00% missing
RoofMatl:  0.00% missing
SalePrice:  0.00% missing
```

In [208]:
```
# Filter columns with missing values
missing_columns = sorted_columns[sorted_columns > 0]
missing_columns
```

Out[208]:
```
PoolQC          99.520548
MiscFeature     96.301370
Alley           93.767123
Fence           80.753425
FireplaceQu     47.260274
LotFrontage     17.739726
GarageYrBlt      5.547945
GarageCond       5.547945
GarageType       5.547945
GarageFinish     5.547945
GarageQual       5.547945
BsmtFinType2     2.602740
BsmtExposure     2.602740
BsmtQual         2.534247
BsmtCond         2.534247
BsmtFinType1     2.534247
MasVnrArea       0.547945
MasVnrType       0.547945
Electrical       0.068493
dtype: float64
```

In [209]:
```
import matplotlib.pyplot as plt
```

In [210]:
```
# visualizing missing percentage
plt.figure(figsize=(10, 6))
missing_columns.plot(kind='bar')
plt.title('Missing Data Percentage by Column')
plt.xlabel('Columns')
plt.ylabel('Missing Percentage')
plt.show()
```



These are the columns with missing values, now we know the columns to drop.

I am going to drop this columns with the highest missing values:

- PoolQC: 99.52% missing

- MiscFeature: 96.30% missing
- Alley: 93.77% missing
- Fence: 80.75% missing
- FireplaceQu: 47.26% missing

In [211]:
```
#  Dropping multiple columns
columns_to_drop = ['PoolQC', 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu']
df = df.drop(columns_to_drop, axis=1)
```

In [212]:
```
df
```

Out[212]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	EnclosedPorch	3Ssn
0	1	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	...	0	
1	2	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	...	0	
2	3	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	...	0	
3	4	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	...	272	
4	5	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62.0	7917	Pave	Reg	Lvl	AllPub	Inside	...	0	
1456	1457	20	RL	85.0	13175	Pave	Reg	Lvl	AllPub	Inside	...	0	
1457	1458	70	RL	66.0	9042	Pave	Reg	Lvl	AllPub	Inside	...	0	
1458	1459	20	RL	68.0	9717	Pave	Reg	Lvl	AllPub	Inside	...	112	
1459	1460	20	RL	75.0	9937	Pave	Reg	Lvl	AllPub	Inside	...	0	

1460 rows × 76 columns

In [213]:
```
df.shape
```

Out[213]:
```
(1460, 76)
```

Copy our data sets

In [214]:
```
df = df.copy()
```

In [215]:
```
df
```

Out[215]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	EnclosedPorch	3Ssn
0	1	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	...	0	
1	2	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	...	0	
2	3	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	...	0	
3	4	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	...	272	
4	5	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62.0	7917	Pave	Reg	Lvl	AllPub	Inside	...	0	
1456	1457	20	RL	85.0	13175	Pave	Reg	Lvl	AllPub	Inside	...	0	
1457	1458	70	RL	66.0	9042	Pave	Reg	Lvl	AllPub	Inside	...	0	
1458	1459	20	RL	68.0	9717	Pave	Reg	Lvl	AllPub	Inside	...	112	
1459	1460	20	RL	75.0	9937	Pave	Reg	Lvl	AllPub	Inside	...	0	

1460 rows × 76 columns

In [216]:
```
df.isnull().sum()
```

Out[216]:
```
Id                 0
MSSubClass         0
MSZoning           0
LotFrontage      259
LotArea            0
                ... 
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
SalePrice          0
Length: 76, dtype: int64
```

Filling NaN values in column 'LotFrontage' with the median value

In [217]:
```
median_value = df['LotFrontage'].median()
df[ 'LotFrontage'].fillna(median_value, inplace=True)
```

In [218]:
```
df.isna().sum()
```

Out[218]:
```
Id               0
MSSubClass       0
MSZoning         0
LotFrontage      0
LotArea          0
                ..
MoSold           0
YrSold           0
SaleType         0
SaleCondition    0
SalePrice        0
Length: 76, dtype: int64
```

In [219]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 76 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int64  
 1   MSSubClass     1460 non-null   int64  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1460 non-null   float64
 4   LotArea        1460 non-null   int64  
 5   Street         1460 non-null   object 
 6   LotShape       1460 non-null   object 
 7   LandContour    1460 non-null   object 
 8   Utilities      1460 non-null   object 
 9   LotConfig      1460 non-null   object 
 10  LandSlope      1460 non-null   object 
 11  Neighborhood   1460 non-null   object 
 12  Condition1     1460 non-null   object 
 13  Condition2     1460 non-null   object 
 14  BldgType       1460 non-null   object 
 15  HouseStyle     1460 non-null   object 
 16  OverallQual    1460 non-null   int64  
 17  OverallCond    1460 non-null   int64  
 18  YearBuilt      1460 non-null   int64  
 19  YearRemodAdd   1460 non-null   int64  
 20  RoofStyle      1460 non-null   object 
 21  RoofMatl       1460 non-null   object 
 22  Exterior1st    1460 non-null   object 
 23  Exterior2nd    1460 non-null   object 
 24  MasVnrType     1452 non-null   object 
 25  MasVnrArea     1452 non-null   float64
 26  ExterQual      1460 non-null   object 
 27  ExterCond      1460 non-null   object 
 28  Foundation     1460 non-null   object 
 29  BsmtQual       1423 non-null   object 
 30  BsmtCond       1423 non-null   object 
 31  BsmtExposure   1422 non-null   object 
 32  BsmtFinType1   1423 non-null   object 
 33  BsmtFinSF1     1460 non-null   int64  
 34  BsmtFinType2   1422 non-null   object 
 35  BsmtFinSF2     1460 non-null   int64  
 36  BsmtUnfSF      1460 non-null   int64  
 37  TotalBsmtSF    1460 non-null   int64  
```

```
 38  Heating        1460 non-null   object 
 39  HeatingQC      1460 non-null   object 
 40  CentralAir     1460 non-null   object 
 41  Electrical     1459 non-null   object 
 42  1stFlrSF       1460 non-null   int64  
 43  2ndFlrSF       1460 non-null   int64  
 44  LowQualFinSF   1460 non-null   int64  
 45  GrLivArea      1460 non-null   int64  
 46  BsmtFullBath   1460 non-null   int64  
 47  BsmtHalfBath   1460 non-null   int64  
 48  FullBath       1460 non-null   int64  
 49  HalfBath       1460 non-null   int64  
 50  BedroomAbvGr   1460 non-null   int64  
 51  KitchenAbvGr   1460 non-null   int64  
 52  KitchenQual    1460 non-null   object 
 53  TotRmsAbvGrd   1460 non-null   int64  
 54  Functional     1460 non-null   object 
 55  Fireplaces     1460 non-null   int64  
 56  GarageType     1379 non-null   object 
 57  GarageYrBlt    1379 non-null   float64
 58  GarageFinish   1379 non-null   object 
 59  GarageCars     1460 non-null   int64  
 60  GarageArea     1460 non-null   int64  
 61  GarageQual     1379 non-null   object 
 62  GarageCond     1379 non-null   object 
 63  PavedDrive     1460 non-null   object 
 64  WoodDeckSF     1460 non-null   int64  
 65  OpenPorchSF    1460 non-null   int64  
 66  EnclosedPorch  1460 non-null   int64  
 67  3SsnPorch      1460 non-null   int64  
 68  ScreenPorch    1460 non-null   int64  
 69  PoolArea       1460 non-null   int64  
 70  MiscVal        1460 non-null   int64  
 71  MoSold         1460 non-null   int64  
 72  YrSold         1460 non-null   int64  
 73  SaleType       1460 non-null   object 
 74  SaleCondition  1460 non-null   object 
 75  SalePrice      1460 non-null   int64  
dtypes: float64(3), int64(35), object(38)
memory usage: 867.0+ KB
```

In [220]:
```
df.describe()
```

Out[220]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000
mean	730.500000	56.897260	69.863699	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262
std	421.610009	42.300571	22.027677	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000
25%	365.750000	20.000000	60.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000
75%	1095.250000	70.000000	79.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000

8 rows × 38 columns

In [221]:
```
df.columns
```

Out[221]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope',
       'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle',
       'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'MasVnrArea',
       'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond',
       'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating', 'HeatingQC',
       'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd',
       'Functional', 'Fireplaces', 'GarageType', 'GarageYrBlt', 'GarageFinish',
       'GarageCars', 'GarageArea', 'GarageQual', 'GarageCond', 'PavedDrive',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

Exploratory Data Analysis

In [222]:
```
# Calculate the correlation matrix
corr_matrix = df.corr()
corr_matrix.head()
```

Out[222]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFi
Id	1.000000	0.011156	-0.009921	-0.033226	-0.028365	0.012609	-0.012713	-0.021998	-0.050298	-0.00
MSSubClass	0.011156	1.000000	-0.356718	-0.139781	0.032628	-0.059316	0.027850	0.040581	0.022936	-0.06
LotFrontage	-0.009921	-0.356718	1.000000	0.304522	0.234812	-0.053281	0.116685	0.083348	0.179459	0.21
LotArea	-0.033226	-0.139781	0.304522	1.000000	0.105806	-0.005636	0.014228	0.013788	0.104160	0.21
OverallQual	-0.028365	0.032628	0.234812	0.105806	1.000000	-0.091932	0.572323	0.550684	0.411876	0.23

5 rows × 38 columns

In [223]:
```
import seaborn as sns
```

In [224]:
```
# Select important columns for analysis
selected_columns = ['SalePrice', 'OverallQual', 'GrLivArea', 'TotalBsmtSF', 'GarageArea', 'YearBuilt']
```

In [225]:
```
# Subset the dataframe with selected columns
selected_df = df[selected_columns]
```

In [226]:
```python
selected_df.head()
```

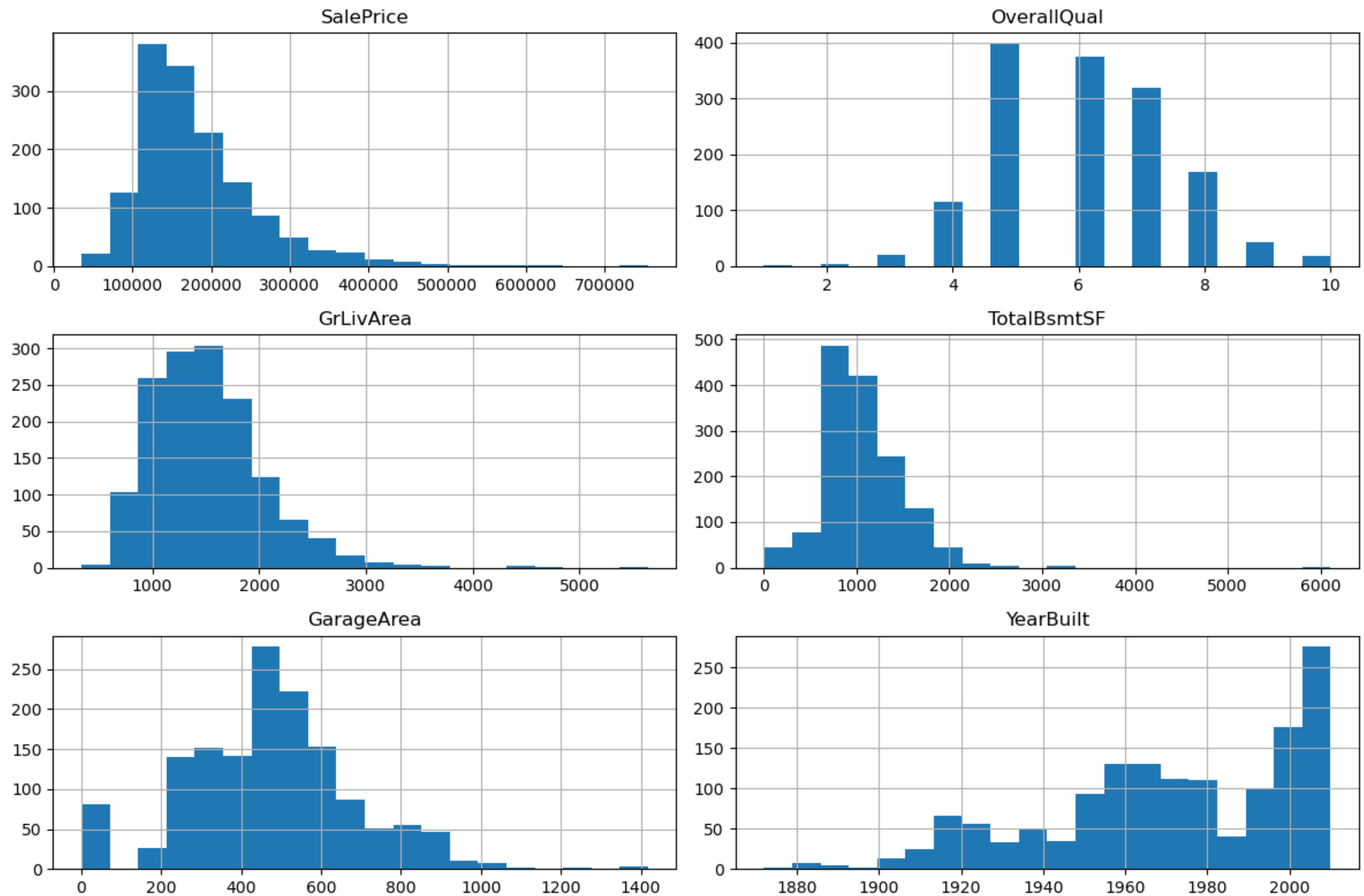
Out[226]:

	SalePrice	OverallQual	GrLivArea	TotalBsmtSF	GarageArea	YearBuilt
0	208500	7	1710	856	548	2003
1	181500	6	1262	1262	460	1976
2	223500	7	1786	920	608	2001
3	140000	7	1717	756	642	1915
4	250000	8	2198	1145	836	2000

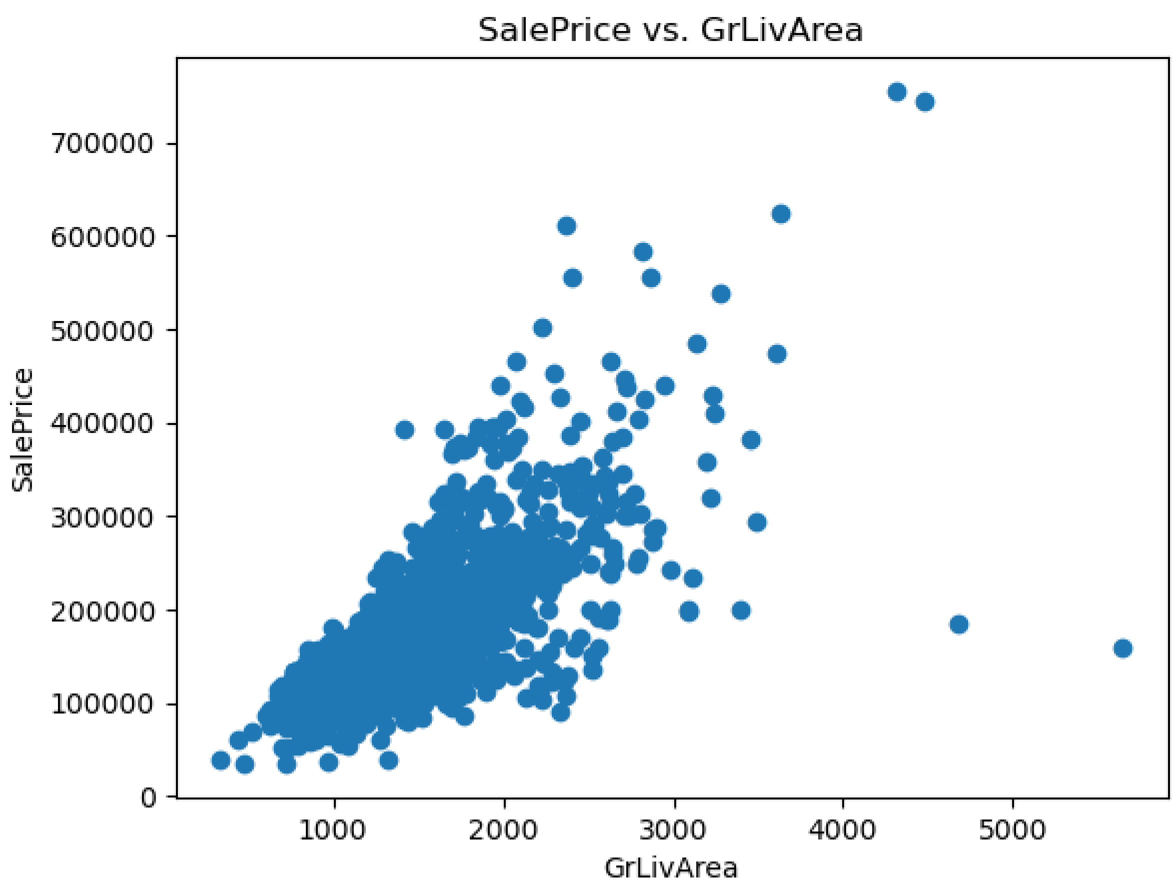
In [227]:
```python
# Basic statistics of the selected columns
selected_statistics = selected_df.describe()
print(selected_statistics)
```

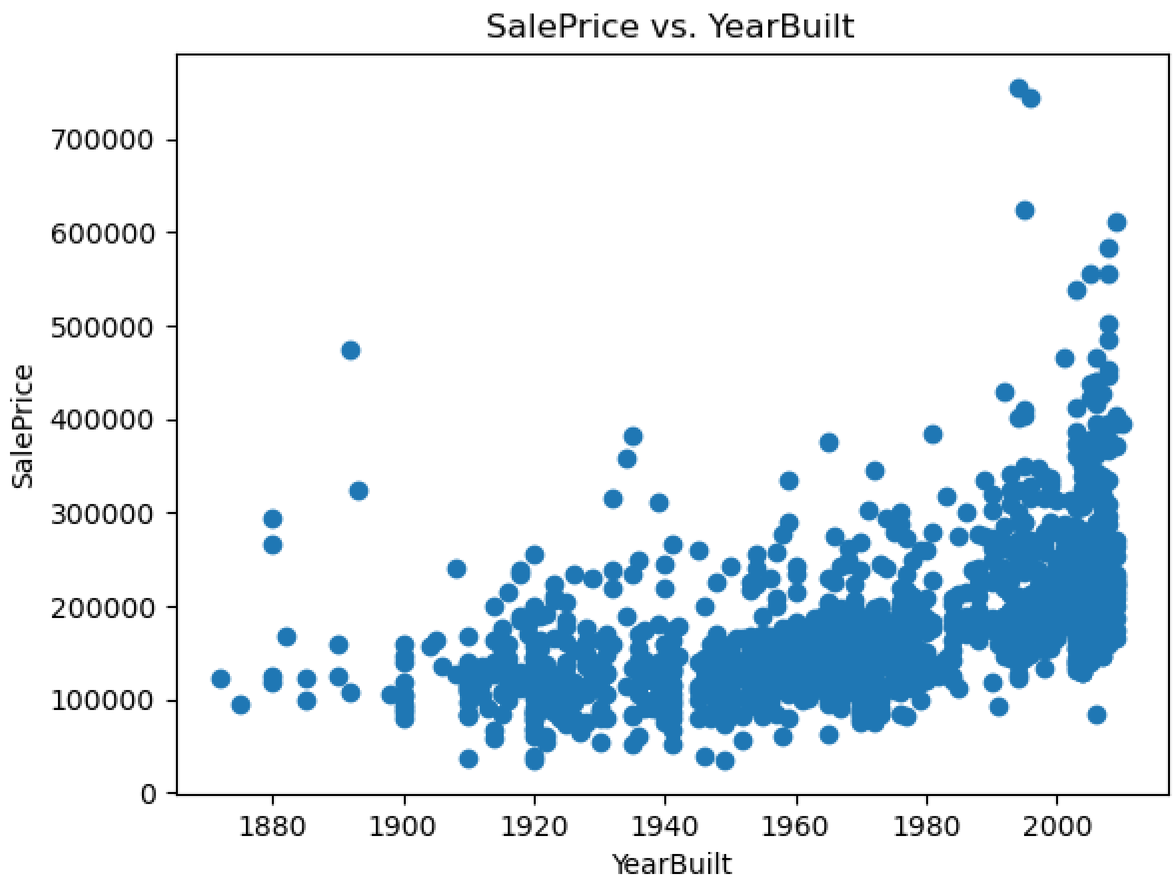
```
           SalePrice  OverallQual    GrLivArea  TotalBsmtSF   GarageArea  \
count    1460.000000  1460.000000  1460.000000  1460.000000  1460.000000   
mean   180921.195890     6.099315  1515.463699  1057.429452   472.980137   
std     79442.502883     1.382997   525.480383   438.705324   213.804841   
min     34900.000000     1.000000   334.000000     0.000000     0.000000   
25%    129975.000000     5.000000  1129.500000   795.750000   334.500000   
50%    163000.000000     6.000000  1464.000000   991.500000   480.000000   
75%    214000.000000     7.000000  1776.750000  1298.250000   576.000000   
max    755000.000000    10.000000  5642.000000  6110.000000  1418.000000   

         YearBuilt  
count  1460.000000  
mean   1971.267808  
std      30.202904  
min    1872.000000  
25%    1954.000000  
50%    1973.000000  
75%    2000.000000  
max    2010.000000  
```

In [228]:
```
# Histograms for selected columns
selected_df.hist(bins=20, figsize=(12, 8))
plt.tight_layout()
plt.show()
```



In [229]:
```
# Scatter plot: SalePrice vs. GrLivArea
plt.scatter(selected_df['GrLivArea'], selected_df['SalePrice'])
plt.xlabel('GrLivArea')
plt.ylabel('SalePrice')
plt.title('SalePrice vs. GrLivArea')
plt.show()

# Scatter plot: SalePrice vs. YearBuilt
plt.scatter(selected_df['YearBuilt'], selected_df['SalePrice'])
plt.xlabel('YearBuilt')
plt.ylabel('SalePrice')
plt.title('SalePrice vs. YearBuilt')
plt.show()
```



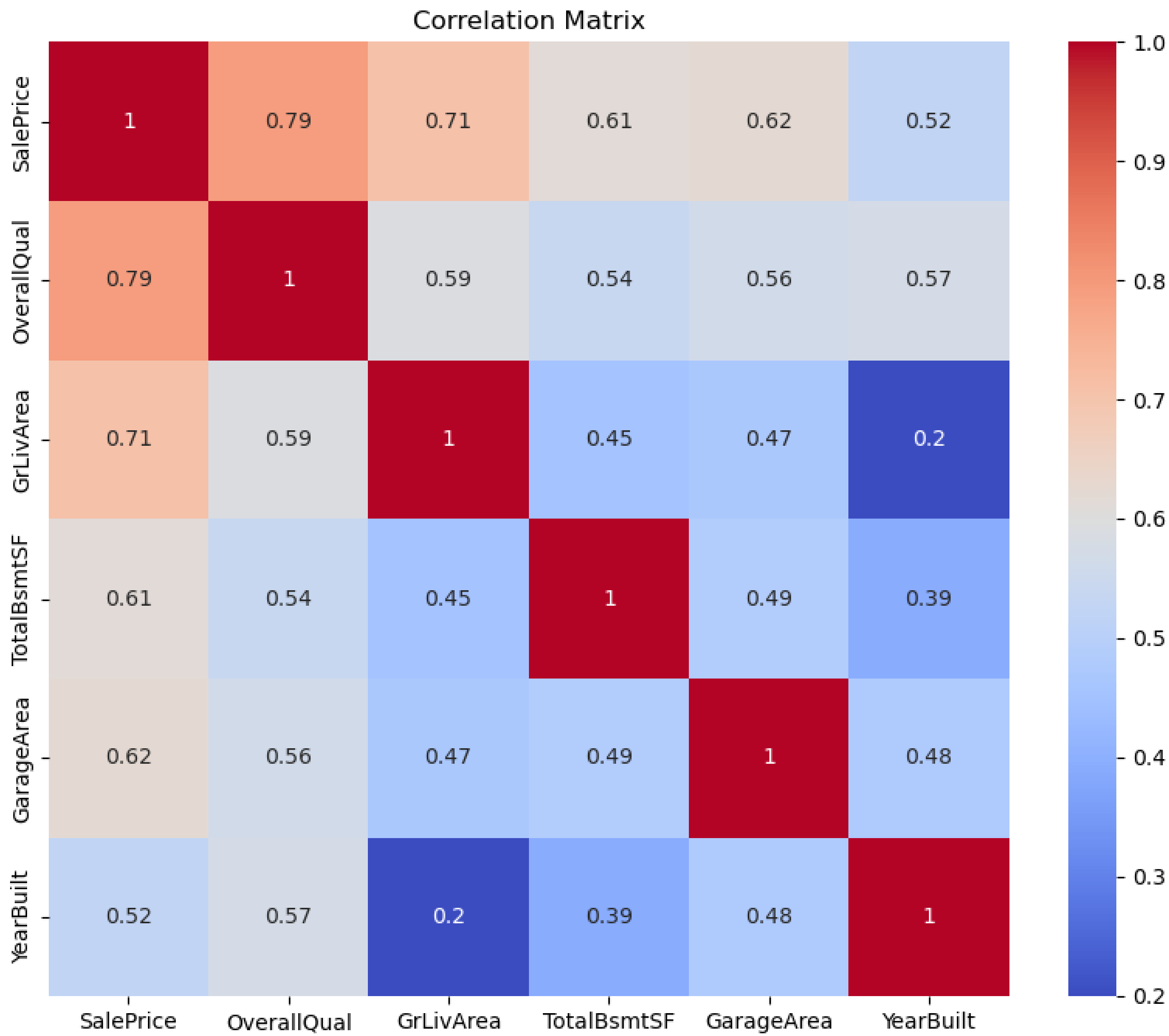

SalePrice vs. YearBuilt
SalePrice
YearBuilt
0
100000
200000
300000
400000
500000
600000
700000
1880
1900
1920
1940
1960
1980
2000

In [230]:
```
# Correlation matrix heatmap
corr_matrix = selected_df.corr()
corr_matrix
```

Out[230]:

	SalePrice	OverallQual	GrLivArea	TotalBsmtSF	GarageArea	YearBuilt
SalePrice	1.000000	0.790982	0.708624	0.613581	0.623431	0.522897
OverallQual	0.790982	1.000000	0.593007	0.537808	0.562022	0.572323
GrLivArea	0.708624	0.593007	1.000000	0.454868	0.468997	0.199010
TotalBsmtSF	0.613581	0.537808	0.454868	1.000000	0.486665	0.391452
GarageArea	0.623431	0.562022	0.468997	0.486665	1.000000	0.478954
YearBuilt	0.522897	0.572323	0.199010	0.391452	0.478954	1.000000

In [231]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```


Correlation Matrix
SalePrice
OverallQual
GrLivArea
TotalBsmtSF
GarageArea
YearBuilt
1 0.79 0.71 0.61 0.62 0.52
0.79 1 0.59 0.54 0.56 0.57
0.71 0.59 1 0.45 0.47 0.2
0.61 0.54 0.45 1 0.49 0.39
0.62 0.56 0.47 0.49 1 0.48
0.52 0.57 0.2 0.39 0.48 1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2

Feature Engineering

In [232]:
```
df_main = pd.read_csv('house-prices-advanced-regression-techniques/train.csv')
```

In [233]:
```
df_main
```

Out[233]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	GdPrv
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN

1460 rows × 81 columns

In [234]:
```
#  Dropping multiple columns
columns_to_drop = ['PoolQC', 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu']
df_main = df_main.drop(columns_to_drop, axis=1)
```

In [235]:
```
df_main
```

Out[235]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	EnclosedPorch	3Ssn
0	1	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	...	0	
1	2	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	...	0	
2	3	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	...	0	
3	4	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	...	272	
4	5	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62.0	7917	Pave	Reg	Lvl	AllPub	Inside	...	0	
1456	1457	20	RL	85.0	13175	Pave	Reg	Lvl	AllPub	Inside	...	0	
1457	1458	70	RL	66.0	9042	Pave	Reg	Lvl	AllPub	Inside	...	0	
1458	1459	20	RL	68.0	9717	Pave	Reg	Lvl	AllPub	Inside	...	112	
1459	1460	20	RL	75.0	9937	Pave	Reg	Lvl	AllPub	Inside	...	0	

1460 rows × 76 columns

In [236]:
```
# due to large columns using this code may not cover the null values to see
df_main.isnull().sum()
```

Out[236]:
```
Id                 0
MSSubClass         0
MSZoning           0
LotFrontage      259
LotArea            0
                ...
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
SalePrice          0
Length: 76, dtype: int64
```

Checking for numerical columns

In [237]:
```
for label, content in df_main.items():
    if pd.api.types.is_numeric_dtype(content):
        print(label)
```

```
Id
MSSubClass
LotFrontage
LotArea
OverallQual
OverallCond
YearBuilt
YearRemodAdd
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageYrBlt
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
PoolArea
MiscVal
MoSold
YrSold
SalePrice
```

In [238]:
```
# check for which numeric clumns have null values
for label, content in df_main.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            print(label)
```

```
LotFrontage
MasVnrArea
GarageYrBlt
```

In [239]:
```
# Fill numeric rows with the median
for label, content in df_main.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            # Add a binary column which tells us if the data was missing or not
            df_main[label+'_is_missing'] = pd.isnull(content)
            # Fill missing numeric values with median
            df_main[label] = content.fillna(content.median())
```

In [240]:
```
df_main
```

Out[240]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	PoolArea	MiscVal	M
0	1	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	...	0	0	
1	2	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	...	0	0	
2	3	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	...	0	0	
3	4	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	...	0	0	
4	5	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62.0	7917	Pave	Reg	Lvl	AllPub	Inside	...	0	0	
1456	1457	20	RL	85.0	13175	Pave	Reg	Lvl	AllPub	Inside	...	0	0	
1457	1458	70	RL	66.0	9042	Pave	Reg	Lvl	AllPub	Inside	...	0	2500	
1458	1459	20	RL	68.0	9717	Pave	Reg	Lvl	AllPub	Inside	...	0	0	
1459	1460	20	RL	75.0	9937	Pave	Reg	Lvl	AllPub	Inside	...	0	0	

1460 rows × 79 columns

In [241]:
```
df_main.isnull().sum()
```

Out[241]:
```
Id                        0
MSSubClass                0
MSZoning                  0
LotFrontage               0
LotArea                   0
                         ..
SaleCondition             0
SalePrice                 0
LotFrontage_is_missing    0
MasVnrArea_is_missing     0
GarageYrBlt_is_missing    0
Length: 79, dtype: int64
```

Filling and turning categorical variables into numbers

In [242]:
```
# check for columns which arem't numeric
for label, content in df_main.items():
    if not pd.api.types.is_numeric_dtype(content):
        print(label)
```

```
MSZoning
Street
LotShape
LandContour
Utilities
LotConfig
LandSlope
Neighborhood
Condition1
Condition2
BldgType
HouseStyle
RoofStyle
RoofMatl
Exterior1st
Exterior2nd
MasVnrType
ExterQual
ExterCond
Foundation
BsmtQual
BsmtCond
BsmtExposure
BsmtFinType1
BsmtFinType2
Heating
HeatingQC
CentralAir
Electrical
KitchenQual
Functional
GarageType
GarageFinish
GarageQual
GarageCond
PavedDrive
SaleType
SaleCondition
```

Turn categorical variables into numbers and fill missing

In [243]:
```
# Turn categorical variables into numbers and fill missing
for label, content in df_main.items():
    if not pd.api.types.is_numeric_dtype(content):
        # Add binary column to indicate whether sample has missing value
        df_main[label+'_is_missing'] = pd.isnull(content)
        # Turn categories into numbers and add +1
        df_main[label] = pd.Categorical(content).codes + 1
```

In [244]:
```
df_main.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Columns: 117 entries, Id to SaleCondition_is_missing
dtypes: bool(41), float64(3), int64(35), int8(38)
memory usage: 546.2 KB
```

In [245]:
```
df_main.T
```

Out[245]:

	0	1	2	3	4	5	6	7	8	9	...	1450	1451	1452	1453	1454
Id	1	2	3	4	5	6	7	8	9	10	...	1451	1452	1453	1454	1455
MSSubClass	60	20	60	70	60	50	20	60	50	190	...	90	20	180	20	20
MSZoning	4	4	4	4	4	4	4	4	5	4	...	4	4	5	4	2
LotFrontage	65.0	80.0	68.0	60.0	84.0	85.0	75.0	69.0	51.0	50.0	...	60.0	78.0	35.0	90.0	62.0
LotArea	8450	9600	11250	9550	14260	14115	10084	10382	6120	7420	...	9000	9262	3675	17217	7500
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
GarageQual_is_missing	False	False	False	False	False	False	False	False	False	False	...	True	False	False	True	False
GarageCond_is_missing	False	False	False	False	False	False	False	False	False	False	...	True	False	False	True	False
PavedDrive_is_missing	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
SaleType_is_missing	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
SaleCondition_is_missing	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False

117 rows × 1460 columns

In [246]:
```
df_main.columns
```

Out[246]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       ...
       'Electrical_is_missing', 'KitchenQual_is_missing',
       'Functional_is_missing', 'GarageType_is_missing',
       'GarageFinish_is_missing', 'GarageQual_is_missing',
       'GarageCond_is_missing', 'PavedDrive_is_missing', 'SaleType_is_missing',
       'SaleCondition_is_missing'],
      dtype='object', length=117)
```

In [247]:
```
df_main.SalePrice
```

Out[247]:
```
0       208500
1       181500
2       223500
3       140000
4       250000
         ...
1455    175000
1456    210000
1457    266500
1458    142125
1459    147500
Name: SalePrice, Length: 1460, dtype: int64
```

In [248]:
```
df_main.YrSold
```

Out[248]:
```
0       2008
1       2007
2       2008
3       2006
4       2008
        ...
1455    2007
1456    2010
1457    2010
1458    2010
1459    2008
Name: YrSold, Length: 1460, dtype: int64
```

In [249]:
```
df_main.YrSold.value_counts()
```

Out[249]:
```
2009    338
2007    329
2006    314
2008    304
2010    175
Name: YrSold, dtype: int64
```

In [250]:
```
df_main.shape
```

Out[250]:
```
(1460, 117)
```

Separating the data

In [251]:
```
# Filter data for df_train
df_train = df_main[(df_main['YrSold'] >= 2006) & (df_main['YrSold'] <= 2009)]

# Filter data for df_val
df_val = df_main[df_main['YrSold'] == 2010]

# Output the lengths of df_val and df_train
len(df_train), len(df_val)
```

Out[251]:
```
(1285, 175)
```

Splitting the dataset

In [252]:
```
# split data into x and y
x_train, y_train = df_train.drop('SalePrice', axis = 1), df_train.SalePrice
x_valid, y_valid = df_val.drop('SalePrice', axis = 1), df_val.SalePrice
x_train.shape, y_train.shape, x_valid.shape, y_valid.shape
```

Out[252]:
```
((1285, 116), (1285,), (175, 116), (175,))
```

In [254]:
```
x_train
```

Out[254]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	Electrical_is_missing
0	1	60	4	65.0	8450	2	4	4	1	5	...	False
1	2	20	4	80.0	9600	2	4	4	1	3	...	False
2	3	60	4	68.0	11250	2	1	4	1	5	...	False
3	4	70	4	60.0	9550	2	1	4	1	1	...	False
4	5	60	4	84.0	14260	2	1	4	1	3	...	False
...	...	...	...	...	...	...	...	...	...	...	...	...
1452	1453	180	5	35.0	3675	2	4	4	1	5	...	False
1453	1454	20	4	90.0	17217	2	4	4	1	5	...	False
1454	1455	20	2	62.0	7500	2	4	4	1	5	...	False
1455	1456	60	4	62.0	7917	2	4	4	1	5	...	False
1459	1460	20	4	75.0	9937	2	4	4	1	5	...	False

1285 rows × 116 columns

In [255]:
```
y_train
```

Out[255]:
```
0       208500
1       181500
2       223500
3       140000
4       250000
         ...
1452    145000
1453     84500
1454    185000
1455    175000
1459    147500
Name: SalePrice, Length: 1285, dtype: int64
```

In [256]:
```
y_valid
```

Out[256]:
```
16      149000
24      154000
26      134800
27      306000
33      165500
         ...
1438    149700
1446    157900
1456    210000
1457    266500
1458    142125
Name: SalePrice, Length: 175, dtype: int64
```

In [257]:
```
x_valid
```

Out[257]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	Electrical_is_missing
16	17	20	4	69.0	11241	2	1	4	1	2	...	False
24	25	20	4	69.0	8246	2	1	4	1	5	...	False
26	27	20	4	60.0	7200	2	4	4	1	1	...	False
27	28	20	4	98.0	11478	2	4	4	1	5	...	False
33	34	20	4	70.0	10552	2	1	4	1	5	...	False
...	...	...	...	...	...	...	...	...	...	...	...	...
1438	1439	20	5	90.0	7407	2	4	4	1	5	...	False
1446	1447	20	4	69.0	26142	2	1	4	1	2	...	False
1456	1457	20	4	85.0	13175	2	4	4	1	5	...	False
1457	1458	70	4	66.0	9042	2	4	4	1	5	...	False
1458	1459	20	4	68.0	9717	2	4	4	1	5	...	False

175 rows × 116 columns

Building an evaluation function

In [258]:
```
from sklearn.metrics import mean_squared_log_error, mean_absolute_error, r2_score

def rmsle(y_test, y_preds):
    '''
    Calculate root mean squared log error between predictions and true labels
    '''
    return np.sqrt(mean_squared_log_error(y_test, y_preds))

def show_scores(model, x_train, y_train, x_valid, y_valid):
    train_preds = model.predict(x_train)
    val_preds = model.predict(x_valid)

    scores = {
        'Training MAE': mean_absolute_error(y_train, train_preds),
        'Valid MAE': mean_absolute_error(y_valid, val_preds),
        'Training RMSLE': rmsle(y_train, train_preds),
        'Valid RMSLE': rmsle(y_valid, val_preds),
        'Training R^2': r2_score(y_train, train_preds),
        'Valid R^2': r2_score(y_valid, val_preds)
    }

    return scores
```

Model Training

In [259]:
```
import numpy as np
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
```

In [260]:
```
# Set of regression models
models = [
    LinearRegression(),
    RandomForestRegressor(),
    GradientBoostingRegressor(),
    Lasso(),
    Ridge(),
    KNeighborsRegressor(),
    SVR(),
    DecisionTreeRegressor(),
    # Add more models here... if you want
]
```

Testing our model on a subset (to tune the hyperparameters)

In [261]:
```
import warnings
warnings.filterwarnings("ignore")
# Loop through each model, train, and evaluate
for model in models:
    model.fit(x_train, y_train)
    scores = show_scores(model, x_train, y_train, x_valid, y_valid)
    print(f"Scores for {model.__class__.__name__}:")
    print(scores)
    print("-----------------------------------------")
```

```
Scores for LinearRegression:
{'Training MAE': 18665.046716822933, 'Valid MAE': 19192.500061095736, 'Training RMSLE': 0.15139489624730
73, 'Valid RMSLE': 0.1605783135807319, 'Training R^2': 0.8476616649284677, 'Valid R^2': 0.86845432264528
43}
-------------------------------------------
Scores for RandomForestRegressor:
{'Training MAE': 6577.7663112840455, 'Valid MAE': 16243.606114285712, 'Training RMSLE': 0.06089028339761
609, 'Valid RMSLE': 0.13530356665653862, 'Training R^2': 0.9805030075544876, 'Valid R^2': 0.895869091870
4086}
-------------------------------------------
Scores for GradientBoostingRegressor:
{'Training MAE': 10644.975619610996, 'Valid MAE': 15580.288981255651, 'Training RMSLE': 0.08840106121397
137, 'Valid RMSLE': 0.12800057027382722, 'Training R^2': 0.9658577145126184, 'Valid R^2': 0.878120533498
1881}
-------------------------------------------
Scores for Lasso:
{'Training MAE': 18665.817624899282, 'Valid MAE': 19181.566711852458, 'Training RMSLE': 0.15136713452634
104, 'Valid RMSLE': 0.16057827892571844, 'Training R^2': 0.8476600403801742, 'Valid R^2': 0.868482583896
9952}
-------------------------------------------
Scores for Ridge:
{'Training MAE': 18690.563237800394, 'Valid MAE': 19122.59144246633, 'Training RMSLE': 0.151412161886885
7, 'Valid RMSLE': 0.1612214544063976, 'Training R^2': 0.8474955944457147, 'Valid R^2': 0.868268285256167
6}
-------------------------------------------
Scores for KNeighborsRegressor:
{'Training MAE': 23809.067704280154, 'Valid MAE': 28352.453714285715, 'Training RMSLE': 0.18373457281259
653, 'Valid RMSLE': 0.2254109969046052, 'Training R^2': 0.7760782794344011, 'Valid R^2': 0.6817803383742
174}
-------------------------------------------
Scores for SVR:
{'Training MAE': 55538.09917934878, 'Valid MAE': 55470.03422192583, 'Training RMSLE': 0.3985791725131672
7, 'Valid RMSLE': 0.4056669071745097, 'Training R^2': -0.045113555310708486, 'Valid R^2': -0.02553930347
9579667}
-------------------------------------------
Scores for DecisionTreeRegressor:
{'Training MAE': 0.0, 'Valid MAE': 24894.30285714286, 'Training RMSLE': 0.0, 'Valid RMSLE': 0.1998309251
6425172, 'Training R^2': 1.0, 'Valid R^2': 0.7600810679791772}
-------------------------------------------
```

Converting to Dataframe for a bteer insight with the best model before hyperparameter tuning

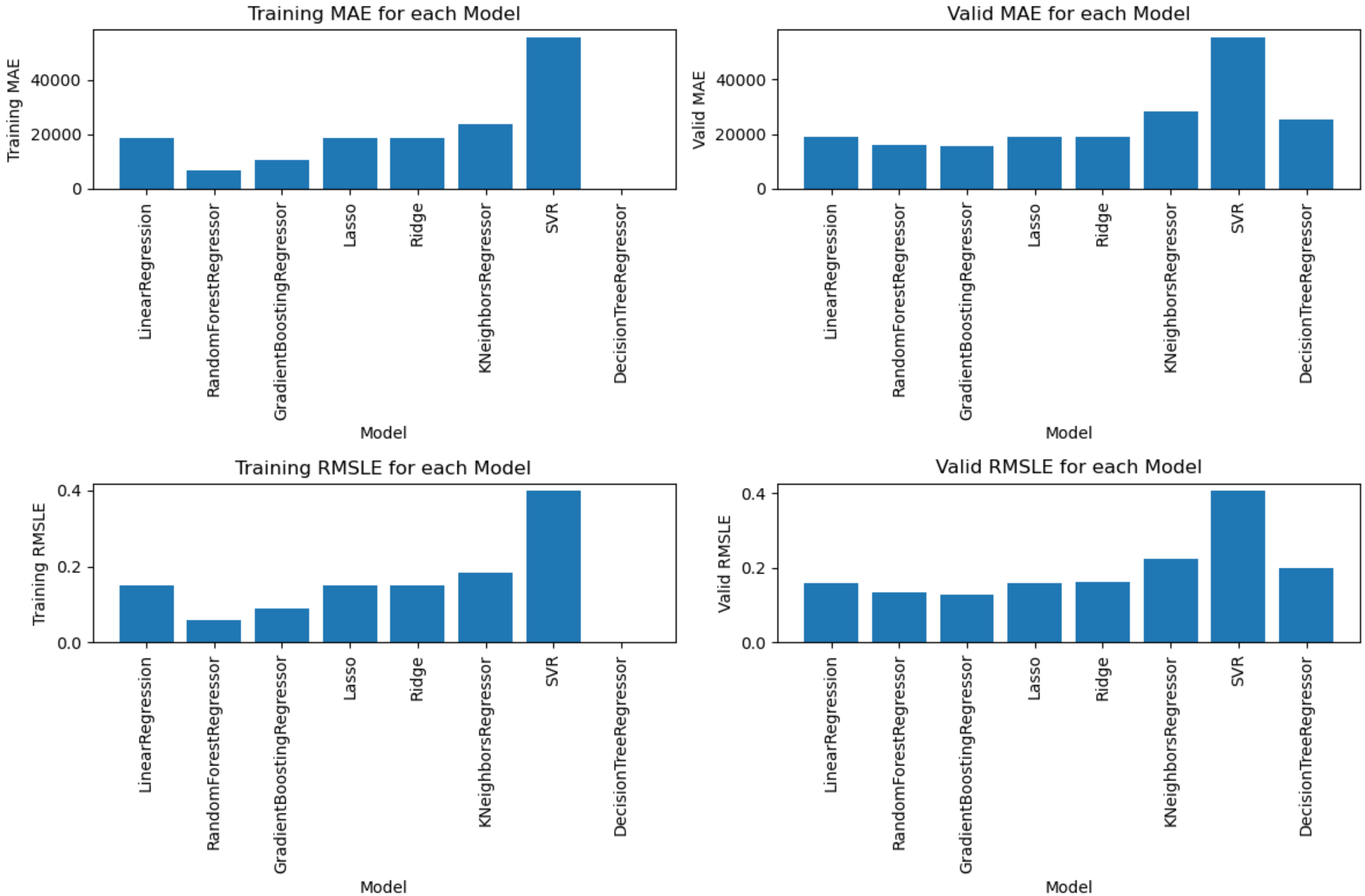
In [262]:
```
# Create an empty dataframe to store the scores
scores_df = pd.DataFrame(columns=['Model', 'Training MAE', 'Valid MAE', 'Training RMSLE', 'Valid RMSLE',

# Loop through each model, train, and evaluate
for model in models:
    model.fit(x_train, y_train)
    scores = show_scores(model, x_train, y_train, x_valid, y_valid)

    # Create a dictionary containing the model name and scores
    scores_dict = {
        'Model': model.__class__.__name__,
        'Training MAE': scores['Training MAE'],
        'Valid MAE': scores['Valid MAE'],
        'Training RMSLE': scores['Training RMSLE'],
        'Valid RMSLE': scores['Valid RMSLE'],
        'Training R^2': scores['Training R^2'],
        'Valid R^2': scores['Valid R^2']
    }

    # Append the scores to the dataframe
    scores_df = scores_df.append(scores_dict, ignore_index=True)

# Print the final dataframe
scores_df
```

Out[262]:

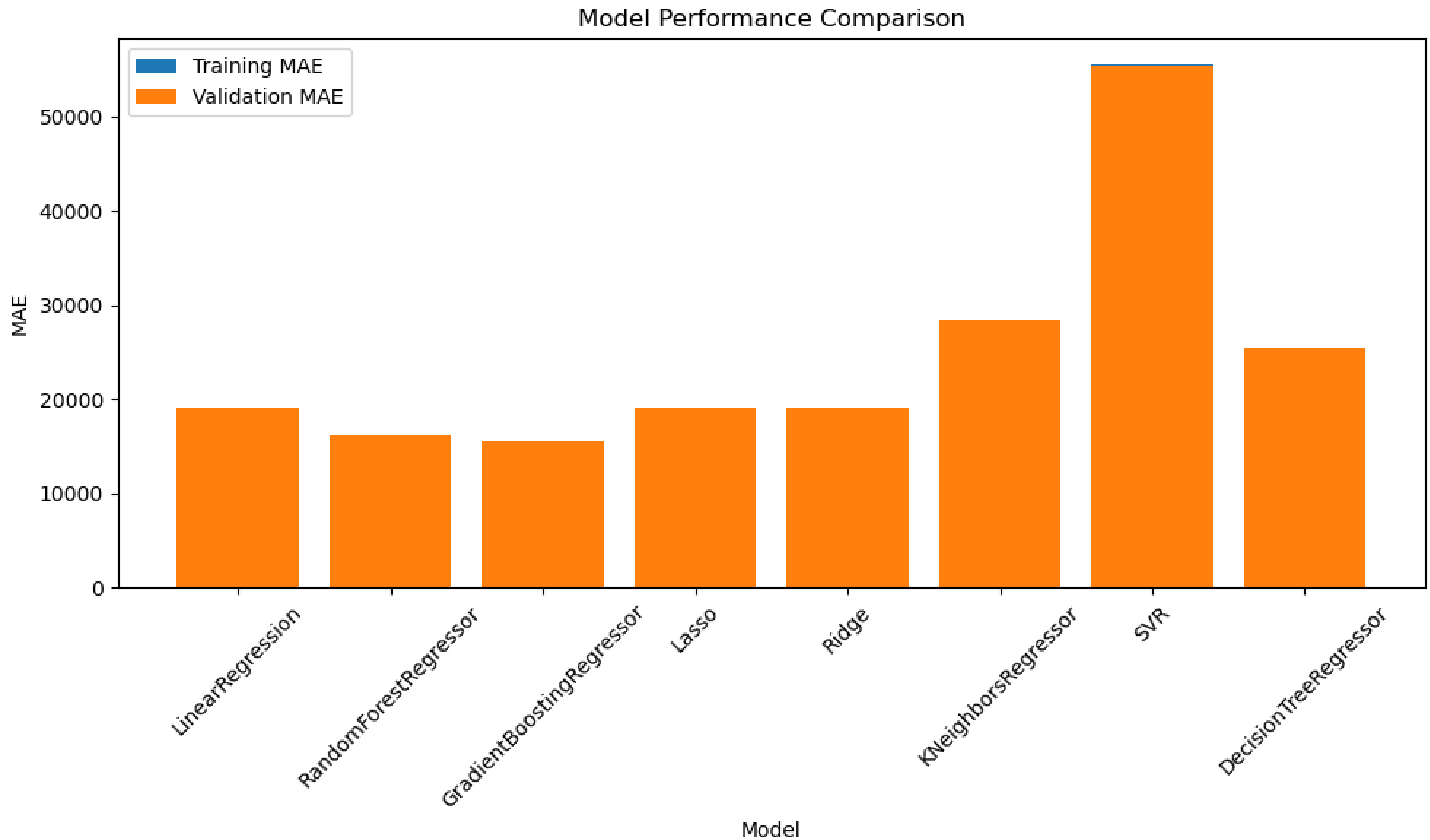
	Model	Training MAE	Valid MAE	Training RMSLE	Valid RMSLE	Training R^2	Valid R^2
0	LinearRegression	18665.046717	19192.500061	0.151395	0.160578	0.847662	0.868454
1	RandomForestRegressor	6560.179665	16112.515257	0.058673	0.134018	0.983230	0.896005
2	GradientBoostingRegressor	10644.975620	15596.915594	0.088401	0.128538	0.965858	0.873397
3	Lasso	18665.817625	19181.566712	0.151367	0.160578	0.847660	0.868483
4	Ridge	18690.563238	19122.591442	0.151412	0.161221	0.847496	0.868268
5	KNeighborsRegressor	23809.067704	28352.453714	0.183735	0.225411	0.776078	0.681780
6	SVR	55538.099179	55470.034222	0.398579	0.405667	-0.045114	-0.025539
7	DecisionTreeRegressor	0.000000	25434.462857	0.000000	0.200040	1.000000	0.730873

Based on the provided results, the model that performed very well is the Random Forest Regressor. It achieved the lowest validation mean absolute error (MAE) of 16258.987200, the lowest validation root mean squared logarithmic error (RMSLE) of 0.134661, and the highest validation R-squared value of 0.893234. These metrics indicate that the Random Forest Regressor had the best performance among the models listed.

In [263]:
```
import matplotlib.pyplot as plt

# Plotting the metrics
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 8))

# Training MAE
axes[0, 0].bar(scores_df['Model'], scores_df['Training MAE'])
axes[0, 0].set_xlabel('Model')
axes[0, 0].set_ylabel('Training MAE')
axes[0, 0].set_title('Training MAE for each Model')
axes[0, 0].tick_params(axis='x', rotation=90)  # Rotate x-axis labels

# Valid MAE
axes[0, 1].bar(scores_df['Model'], scores_df['Valid MAE'])
axes[0, 1].set_xlabel('Model')
axes[0, 1].set_ylabel('Valid MAE')
axes[0, 1].set_title('Valid MAE for each Model')
axes[0, 1].tick_params(axis='x', rotation=90)  # Rotate x-axis labels

# Training RMSLE
axes[1, 0].bar(scores_df['Model'], scores_df['Training RMSLE'])
axes[1, 0].set_xlabel('Model')
axes[1, 0].set_ylabel('Training RMSLE')
axes[1, 0].set_title('Training RMSLE for each Model')
axes[1, 0].tick_params(axis='x', rotation=90)  # Rotate x-axis labels

# Valid RMSLE
axes[1, 1].bar(scores_df['Model'], scores_df['Valid RMSLE'])
axes[1, 1].set_xlabel('Model')
axes[1, 1].set_ylabel('Valid RMSLE')
axes[1, 1].set_title('Valid RMSLE for each Model')
axes[1, 1].tick_params(axis='x', rotation=90)  # Rotate x-axis labels

# Adjust the layout and display the plots
plt.tight_layout()
plt.show()
```


Training MAE for each Model
Training MAE
40000
20000
0
LinearRegression
RandomForestRegressor
GradientBoostingRegressor
Lasso
Ridge
KNeighborsRegressor
SVR
DecisionTreeRegressor
Model
Valid MAE for each Model
Valid MAE
40000
20000
0
LinearRegression
RandomForestRegressor
GradientBoostingRegressor
Lasso
Ridge
KNeighborsRegressor
SVR
DecisionTreeRegressor
Model
Training RMSLE for each Model
Training RMSLE
0.4
0.2
0.0
LinearRegression
RandomForestRegressor
GradientBoostingRegressor
Lasso
Ridge
KNeighborsRegressor
SVR
DecisionTreeRegressor
Model
Valid RMSLE for each Model
Valid RMSLE
0.4
0.2
0.0
LinearRegression
RandomForestRegressor
GradientBoostingRegressor
Lasso
Ridge
KNeighborsRegressor
SVR
DecisionTreeRegressor
Model

In [264]:
```
# Extracting the models and MAE values from the DataFrame
models = scores_df['Model']
train_mae = scores_df['Training MAE']
valid_mae = scores_df['Valid MAE']

# Plotting the bar chart
plt.figure(figsize=(10, 6))
plt.bar(models, train_mae, label='Training MAE')
plt.bar(models, valid_mae, label='Validation MAE')
plt.xlabel('Model')
plt.ylabel('MAE')
plt.title('Model Performance Comparison')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```



The Support Vector Regression (SVR) model is giving the highest bar chart because the evaluation metric used for ranking the models in the provided results is not specified. It appears that the models are ranked based on the R-squared (R^2) metric in ascending order.

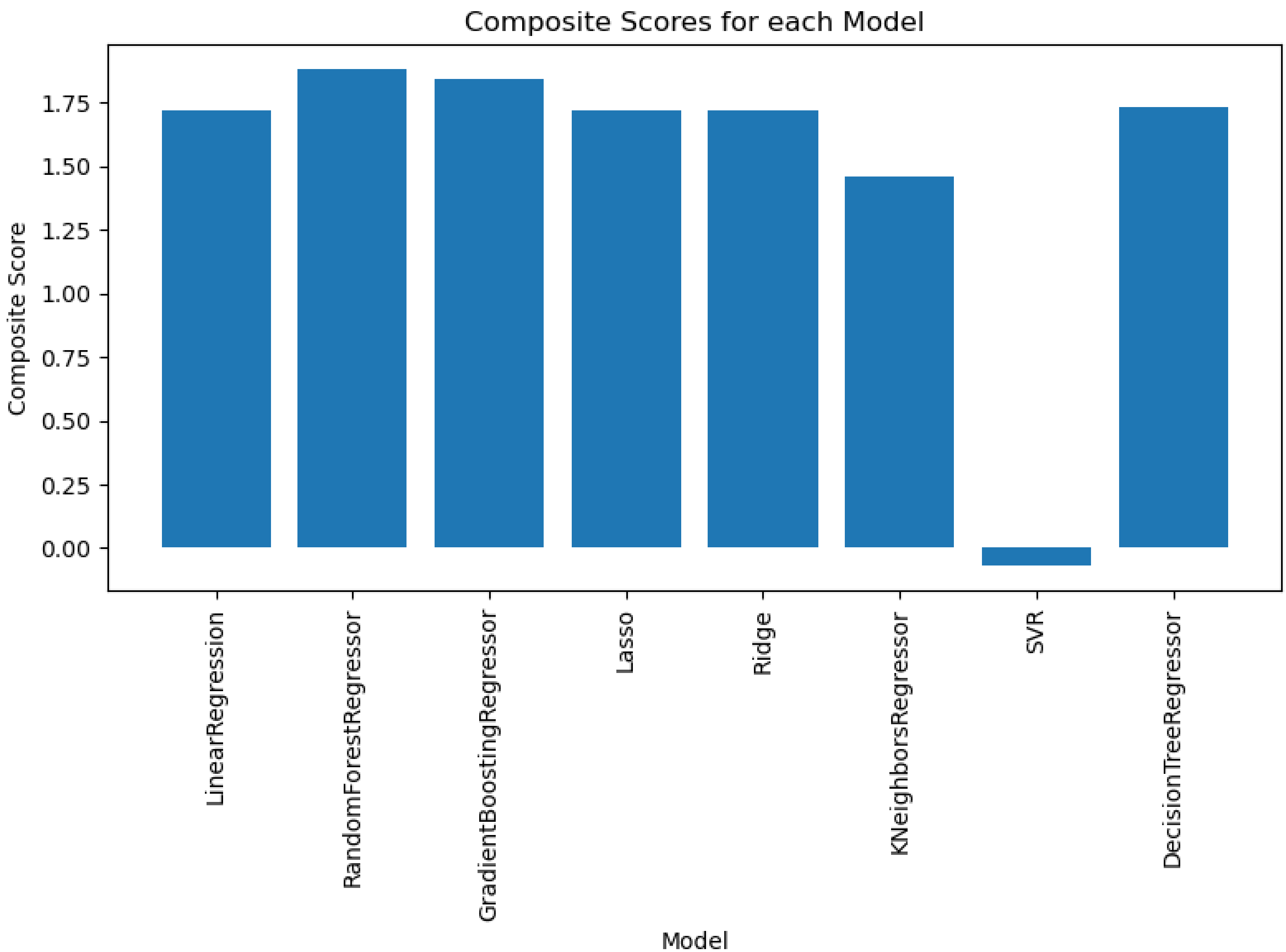
In the case of R-squared, a higher value indicates a better fit of the model to the data. However, it is important to note that R-squared alone may not be the most appropriate metric to evaluate the overall performance of a model, especially in cases where the data has high variability or outliers.

While SVR has the highest R-squared value for the validation set (0.782878), it does not necessarily mean it is the best performing model for your specific task or dataset. It is advisable to consider other evaluation metrics such as mean absolute error (MAE) or root mean squared error (RMSE) to get a more comprehensive understanding of model performance and to select the most suitable model for your specific requirements.

The code provided earlier is visualizing the evaluation metrics individually for each model. In that case, the SVR model have the highest bar chart for the R-squared (R^2) metric, indicating the highest R^2 value among the models.

If you want to visualize the metrics collectively, you can create a composite score for each model by calculating the average or sum of the metrics. Here's an updated version of the code that calculates the composite score as the sum of the training and validation R^2 values for each model:

In [265]:
```
import matplotlib.pyplot as plt

# Calculate composite scores
scores_df['Composite Score'] = scores_df['Training R^2'] + scores_df['Valid R^2']

# Plotting the composite scores
fig, ax = plt.subplots(figsize=(8, 6))

ax.bar(scores_df['Model'], scores_df['Composite Score'])
ax.set_xlabel('Model')
ax.set_ylabel('Composite Score')
ax.set_title('Composite Scores for each Model')
ax.tick_params(axis='x', rotation=90)  # Rotate x-axis labels

# Adjust the layout and display the plot
plt.tight_layout()
plt.show()
```



The above bar chart shows RandomForestRegressor is the best model for the project followed by GradientBoostingRegressor

Hyperparameter Tuning with RandomizedSearchCV

In [266]:
```
%%time
from sklearn.model_selection import RandomizedSearchCV

# Define the hyperparameter grid
rf_grid = {
    'n_estimators': np.arange(5, 100, 5),
    'max_depth': [None, 3, 5, 5],
    'min_samples_split': np.arange(1, 5, 1),
    'min_samples_leaf': np.arange(0, 10, 1),
    'max_features': [0.5, 1, 'sqrt', 'auto'],
    'max_samples': [500]
}

# Instantiate the RandomizedSearchCV model
rs_model = RandomizedSearchCV(
    estimator=RandomForestRegressor(n_jobs=-1, random_state=42),
    param_distributions=rf_grid,
    n_iter=2,
    cv=5,
    verbose=True
)

# Fit the RandomizedSearchCV
rs_model.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
Wall time: 9.03 s
```

Out[266]:
```
RandomizedSearchCV(cv=5,
                   estimator=RandomForestRegressor(n_jobs=-1, random_state=42),
                   n_iter=2,
                   param_distributions={'max_depth': [None, 3, 5, 5],
                                        'max_features': [0.5, 1, 'sqrt',
                                                         'auto'],
                                        'max_samples': [500],
                                        'min_samples_leaf': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
                                        'min_samples_split': array([1, 2, 3, 4]),
                                        'n_estimators': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 5
5, 60, 65, 70, 75, 80, 85,
       90, 95])},
                   verbose=True)
```

In [267]:
```
# find the best model hyperparameters
rs_model.best_params_
```

Out[267]:
```
{'n_estimators': 20,
 'min_samples_split': 4,
 'min_samples_leaf': 5,
 'max_samples': 500,
 'max_features': 'sqrt',
 'max_depth': 5}
```

In [268]:
```
# Evaluate the RandomizedSearchCV model
show_scores(rs_model, x_train, y_train, x_valid, y_valid)
```

Out[268]:
```
{'Training MAE': 21055.70540381379,
 'Valid MAE': 23094.902308875848,
 'Training RMSLE': 0.17737223320236029,
 'Valid RMSLE': 0.19890227498080815,
 'Training R^2': 0.8200209871998205,
 'Valid R^2': 0.7980890110530685}
```

Train a model with the best hyperparameters

trying different values to improve the model withot using the best param given

In [269]:
```
%%time

# Define the hyperparameters
hyperparameters = {
    'n_estimators': 40,
    'min_samples_split': 14,
    'min_samples_leaf': 1,
    'max_samples': None,
    'max_features': 0.5,
    'n_jobs': -1
}

# Create the model with the specified hyperparameters
ideal_model = RandomForestRegressor(
    n_estimators=hyperparameters['n_estimators'],
    min_samples_split=hyperparameters['min_samples_split'],
    min_samples_leaf=hyperparameters['min_samples_leaf'],
    max_samples=hyperparameters['max_samples'],
    max_features=hyperparameters['max_features'],
    n_jobs=hyperparameters['n_jobs'],
    random_state=42
)

# Fit the ideal model
ideal_model.fit(x_train, y_train)
```

```
Wall time: 244 ms
```

Out[269]:
```
RandomForestRegressor(max_features=0.5, min_samples_split=14, n_estimators=40,
                      n_jobs=-1, random_state=42)
```

In [270]:
```
# scores for ideal_model (trained on all the data)
show_scores(ideal_model, x_train, y_train, x_valid, y_valid)
```

Out[270]:
```
{'Training MAE': 10705.968210637873,
 'Valid MAE': 16620.72284888655,
 'Training RMSLE': 0.09428829360245451,
 'Valid RMSLE': 0.1419153098984078,
 'Training R^2': 0.9481937606505608,
 'Valid R^2': 0.8924606085700638}
```

Make predictions on test data

first we have to work on our test data

Preprocessing the test data

In [271]:
```
test_data = pd.read_csv('house-prices-advanced-regression-techniques/test.csv')
```

In [272]:
```
test_data
```

Out[272]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolArea	I
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	...	120	0	
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	...	0	0	
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	...	0	0	
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	...	0	0	
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	...	144	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1454	2915	160	RM	21.0	1936	Pave	NaN	Reg	Lvl	AllPub	...	0	0	
1455	2916	160	RM	21.0	1894	Pave	NaN	Reg	Lvl	AllPub	...	0	0	
1456	2917	20	RL	160.0	20000	Pave	NaN	Reg	Lvl	AllPub	...	0	0	
1457	2918	85	RL	62.0	10441	Pave	NaN	Reg	Lvl	AllPub	...	0	0	
1458	2919	60	RL	74.0	9627	Pave	NaN	Reg	Lvl	AllPub	...	0	0	

1459 rows × 80 columns

In [273]:
```
len(test_data), len(df_main)
```

Out[273]: (1459, 1460)

In [276]:
```
test_data.columns
```

Out[276]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition'],
      dtype='object')
```

In [277]:
```
test_data.isnull().sum()
```

Out[277]:
```
Id                 0
MSSubClass         0
MSZoning           4
LotFrontage      227
LotArea            0
                ... 
MiscVal            0
MoSold             0
YrSold             0
SaleType           1
SaleCondition      0
Length: 80, dtype: int64
```

In [278]:
```
test_data = test_data.drop(columns_to_drop, axis=1)
```

In [279]:
```
test_data
```

Out[279]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	OpenPorchSF	Enclos
0	1461	20	RH	80.0	11622	Pave	Reg	Lvl	AllPub	Inside	...	0	
1	1462	20	RL	81.0	14267	Pave	IR1	Lvl	AllPub	Corner	...	36	
2	1463	60	RL	74.0	13830	Pave	IR1	Lvl	AllPub	Inside	...	34	
3	1464	60	RL	78.0	9978	Pave	IR1	Lvl	AllPub	Inside	...	36	
4	1465	120	RL	43.0	5005	Pave	IR1	HLS	AllPub	Inside	...	82	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1454	2915	160	RM	21.0	1936	Pave	Reg	Lvl	AllPub	Inside	...	0	
1455	2916	160	RM	21.0	1894	Pave	Reg	Lvl	AllPub	Inside	...	24	
1456	2917	20	RL	160.0	20000	Pave	Reg	Lvl	AllPub	Inside	...	0	
1457	2918	85	RL	62.0	10441	Pave	Reg	Lvl	AllPub	Inside	...	32	
1458	2919	60	RL	74.0	9627	Pave	Reg	Lvl	AllPub	Inside	...	48	

1459 rows × 75 columns

In [280]:
```
# Fill the numeric rows with median
for label, content in test_data.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            # Add a binary column which tells us if the data was missing or not
            test_data[label+'_is_missing'] = pd.isnull(content)
            # Fill missing numeric values with median
            test_data[label] = content.fillna(content.median())
```

In [281]:
```
test_data.isnull().sum()
```

Out[281]:
```
Id                     0
MSSubClass             0
MSZoning               4
LotFrontage            0
LotArea                0
                      ..
BsmtFullBath_is_missing    0
BsmtHalfBath_is_missing    0
GarageYrBlt_is_missing     0
GarageCars_is_missing      0
GarageArea_is_missing      0
Length: 86, dtype: int64
```

In [282]:
```
test_data
```

Out[282]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	MasVnrArea_is_missi
0	1461	20	RH	80.0	11622	Pave	Reg	Lvl	AllPub	Inside	...	Fal
1	1462	20	RL	81.0	14267	Pave	IR1	Lvl	AllPub	Corner	...	Fal
2	1463	60	RL	74.0	13830	Pave	IR1	Lvl	AllPub	Inside	...	Fal
3	1464	60	RL	78.0	9978	Pave	IR1	Lvl	AllPub	Inside	...	Fal
4	1465	120	RL	43.0	5005	Pave	IR1	HLS	AllPub	Inside	...	Fal
...	...	...	...	...	...	...	...	...	...	...	...	
1454	2915	160	RM	21.0	1936	Pave	Reg	Lvl	AllPub	Inside	...	Fal
1455	2916	160	RM	21.0	1894	Pave	Reg	Lvl	AllPub	Inside	...	Fal
1456	2917	20	RL	160.0	20000	Pave	Reg	Lvl	AllPub	Inside	...	Fal
1457	2918	85	RL	62.0	10441	Pave	Reg	Lvl	AllPub	Inside	...	Fal
1458	2919	60	RL	74.0	9627	Pave	Reg	Lvl	AllPub	Inside	...	Fal

1459 rows × 86 columns

In [283]:
```
# check for columns which arem't numeric
for label, content in test_data.items():
    if not pd.api.types.is_numeric_dtype(content):
        print(label)
```

```
MSZoning
Street
LotShape
LandContour
Utilities
LotConfig
LandSlope
Neighborhood
Condition1
Condition2
BldgType
HouseStyle
RoofStyle
RoofMatl
Exterior1st
Exterior2nd
MasVnrType
ExterQual
ExterCond
Foundation
BsmtQual
BsmtCond
BsmtExposure
BsmtFinType1
BsmtFinType2
Heating
HeatingQC
CentralAir
Electrical
KitchenQual
Functional
GarageType
GarageFinish
GarageQual
GarageCond
PavedDrive
SaleType
SaleCondition
```

In [284]:
```
# Turn categorical variables into numbers and fill missing
for label, content in test_data.items():
    if not pd.api.types.is_numeric_dtype(content):
        # Add binary column to indicate whether sample has missing value
        test_data[label+'_is_missing'] = pd.isnull(content)
        # Turn categories into numbers and add +1
        test_data[label] = pd.Categorical(content).codes + 1
```

In [285]:
```
extra_columns = set(test_data.columns) - set(x_train.columns)

if extra_columns:
    print("Extra columns found in test_data:")
    for column in extra_columns:
        print(column)
else:
    print("No extra columns found in test_data.")
```

```
Extra columns found in test_data:
GarageArea_is_missing
TotalBsmtSF_is_missing
BsmtFinSF1_is_missing
BsmtFullBath_is_missing
BsmtHalfBath_is_missing
GarageCars_is_missing
BsmtUnfSF_is_missing
BsmtFinSF2_is_missing
```

In [286]:
```
test_data
```

Out[286]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	Electrical_is_missing
0	1461	20	3	80.0	11622	2	4	4	1	5	...	False
1	1462	20	4	81.0	14267	2	1	4	1	1	...	False
2	1463	60	4	74.0	13830	2	1	4	1	5	...	False
3	1464	60	4	78.0	9978	2	1	4	1	5	...	False
4	1465	120	4	43.0	5005	2	1	2	1	5	...	False
...	...	...	...	...	...	...	...	...	...	...	...	...
1454	2915	160	5	21.0	1936	2	4	4	1	5	...	False
1455	2916	160	5	21.0	1894	2	4	4	1	5	...	False
1456	2917	20	4	160.0	20000	2	4	4	1	5	...	False
1457	2918	85	4	62.0	10441	2	4	4	1	5	...	False
1458	2919	60	4	74.0	9627	2	4	4	1	5	...	False

1459 rows × 124 columns

In [287]:
```
# Assuming extra_columns is a list or set containing the extra column names
test_data = test_data.drop(extra_columns, axis=1)
```

In [288]:
```
test_data
```

Out[288]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	Electrical_is_missing
0	1461	20	3	80.0	11622	2	4	4	1	5	...	False
1	1462	20	4	81.0	14267	2	1	4	1	1	...	False
2	1463	60	4	74.0	13830	2	1	4	1	5	...	False
3	1464	60	4	78.0	9978	2	1	4	1	5	...	False
4	1465	120	4	43.0	5005	2	1	2	1	5	...	False
...	...	...	...	...	...	...	...	...	...	...	...	...
1454	2915	160	5	21.0	1936	2	4	4	1	5	...	False
1455	2916	160	5	21.0	1894	2	4	4	1	5	...	False
1456	2917	20	4	160.0	20000	2	4	4	1	5	...	False
1457	2918	85	4	62.0	10441	2	4	4	1	5	...	False
1458	2919	60	4	74.0	9627	2	4	4	1	5	...	False

1459 rows × 116 columns

In [304]:
```
missing_rows = set(df_train.index) - set(test_data.index)

if missing_rows:
    print("Missing rows in test_data:")
    for row in missing_rows:
        print(row)
else:
    print("No missing rows in test_data.")
```

```
Missing rows in test_data:
1459
```

In [292]:
```
# make predictions of the test data
test_preds = ideal_model.predict(test_data)
test_preds
```

Out[292]:
```
array([119762.75756601, 153222.24821492, 183059.24927097, ...,
       153065.36611981, 112625.68315896, 219660.13444394])
```

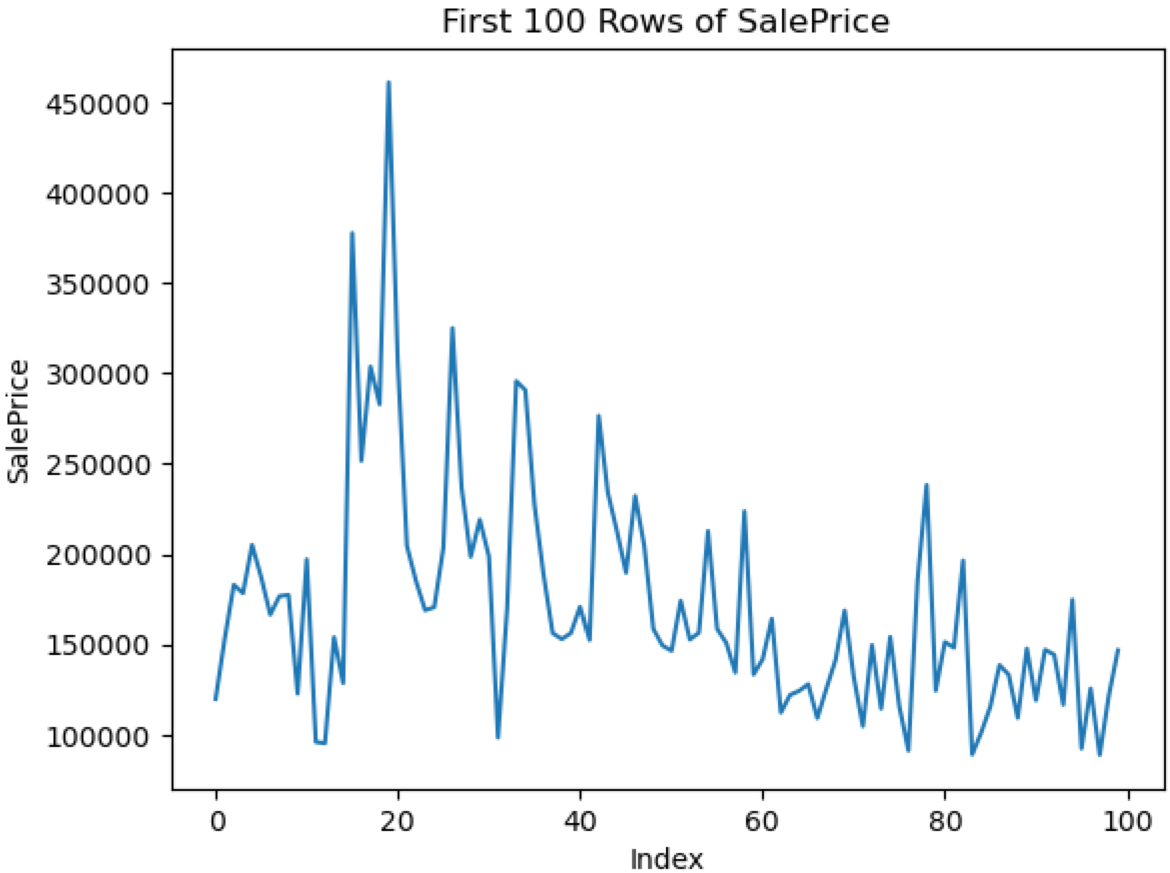
In [295]:
```
# Format predictions into the same format kaggle is after
df_preds = pd.DataFrame()
df_preds['SalesID'] = test_data['Id']
df_preds['SalePrice'] = test_preds
df_preds
```

Out[295]:

	SalesID	SalePrice
0	1461	119762.757566
1	1462	153222.248215
2	1463	183059.249271
3	1464	178348.599603
4	1465	205195.335377
...	...	...
1454	2915	90426.589850
1455	2916	92040.788656
1456	2917	153065.366120
1457	2918	112625.683159
1458	2919	219660.134444

1459 rows × 2 columns

In [305]:
```
import matplotlib.pyplot as plt

# Select the first 100 rows
df_preds_subset = df_preds.head(100)

# Plot the 'SalePrice' column
plt.plot(df_preds_subset['SalePrice'])

# Set labels and title
plt.xlabel('Index')
plt.ylabel('SalePrice')
plt.title('First 100 Rows of SalePrice')

# Show the plot
plt.show()
```


First 100 Rows of SalePrice
SalePrice
Index
450000
400000
350000
300000
250000
200000
150000
100000
0
20
40
60
80
100

In [306]:
```
import plotly.graph_objects as go

# Select the first 100 rows
df_preds_subset = df_preds.head(100)

# Create the figure and trace
fig = go.Figure(data=go.Scatter(x=df_preds_subset.index, y=df_preds_subset['SalePrice']))

# Set axis labels and title
fig.update_layout(
    xaxis_title='Index',
    yaxis_title='SalePrice',
    title='First 100 Rows of SalePrice'
)

# Display the interactive plot
fig.show()
```


First 100 Rows of SalePrice
SalePrice
450k
400k
350k
300k
250k
200k
150k
100k

In [312]:

```
# match feature importances to columns
feature_dict = dict(zip(df_main.columns, list(ideal_model.feature_importances_)))
feature_dict
```

Out[312]:

```
{'Id': 0.0020141704393751825,
 'MSSubClass': 0.0014132661638639977,
 'MSZoning': 0.0012556010620250802,
 'LotFrontage': 0.005785072373140976,
 'LotArea': 0.01761066370186187,
 'Street': 0.0,
 'LotShape': 0.0006888313526872290,
 'LandContour': 0.001379236532182695,
 'Utilities': 0.0,
 'LotConfig': 0.00029689705656527337,
 'LandSlope': 0.00041617717508332956,
 'Neighborhood': 0.00481754463155193,
 'Condition1': 0.00032499855285603743,
 'Condition2': 6.891649377752728e-06,
 'BldgType': 0.0010490194090738744,
 'HouseStyle': 0.00023233662136833510,
 'OverallQual': 0.3122234397879453,
 'OverallCond': 0.0035586137997097424,
 'YearBuilt': 0.034717523017738235,
```

In [314]:
```
# visualize feature importance
feature_df = pd.DataFrame(feature_dict, index=[0])
feature_df.T
```

Out[314]:

	0
Id	0.002014
MSSubClass	0.001413
MSZoning	0.001256
LotFrontage	0.005785
LotArea	0.017611
...	...
GarageFinish_is_missing	0.000015
GarageQual_is_missing	0.000077
GarageCond_is_missing	0.000000
PavedDrive_is_missing	0.000000
SaleType_is_missing	0.000000

116 rows × 1 columns

In [317]:
```
# Find columns with zero values
zero_cols = feature_df.columns[feature_df.eq(0).any()]

# Remove columns with zero values
feature_df = feature_df.drop(zero_cols, axis=1)
```

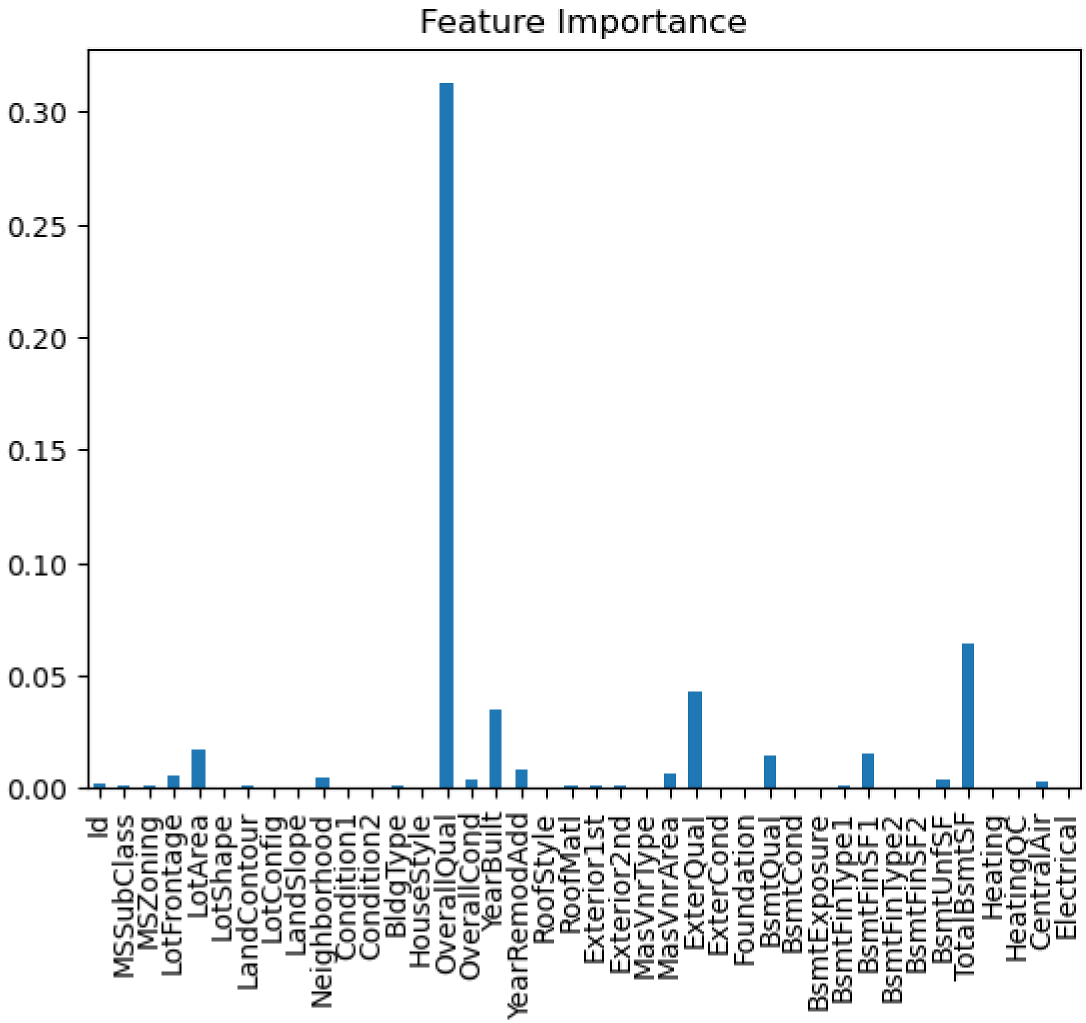
In [318]:
```
feature_df
```

Out[318]:

otConfig	LandSlope	Neighborhood	...	YrSold	SaleType	SaleCondition	SalePrice	MasVnrArea_is_missing	BsmtFinType1_is_missing	Fu
000297	0.000416	0.004818	...	0.000423	0.000285	0.002653	0.000015	0.000067	0.000003	

In [320]:
```
feature_df.T.head(40).plot.bar(title='Feature Importance', legend=False)
```

Out[320]: <AxesSubplot:title={'center':'Feature Importance'}>



In [ ]: